42390.P9918

Patent

UNITED STATES PATENT APPLICATION

FOR

Dynamically Loading Program Code Over A Push-Based Network

INVENTOR:

Jason Hallford

Prepared by

Steven D. Yates
Reg. No. 42,242
(503) 264-6589

Express Mail mailing label number: EL034437271US

Dynamically Loading Program Code Over A Push-Based Network

Field of the Invention

The invention generally relates to pushing programming code over unidirectional communication links, and more particularly, to pushing dynamically loadable program code, such as an object-oriented class definition, over the unidirectional communication link.

Background

FIG. 1 illustrates a general prior art hardware environment for pushing data to a receiving computing device.

As illustrated, there is a data-pushing device **100**, such as one or more network servers or other computing devices, that pushes data onto a network **102**. The data-pushing device is responsible for generating or forwarding data that is ultimately received by a receiving computing device **104**. The network can be any combination of conventional and proprietary networks such as an intranet or the Internet.

A receiving computing device **104**, also in communication with the network, receives the pushed data. Various protocols for receiving pushed data are known in the art. For example, see the PointCast system by EntryPoint of San Diego, California. Generally, the receiving computing device **104** listens to a particular data channel, e.g., a Transmission Control Protocol/Internet Protocol (TCP/IP) network port, broadcast channel, frequency range, etc. for data pushed by the data-pushing device **100**.

Brief Description Of The Drawings

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

FIG. 1 illustrates a general prior-art hardware environment for pushing data to a receiving computing device.

FIG. 2 illustrates a variation on the FIG. 1 push environment, in which the network includes wireless networks.

FIG. 3 is a flowchart, according to one embodiment of the invention, for pushing programming code over a unidirectional communication link.

FIG. 4 illustrates a receiver receiving pushed data in accordance with the FIG. 3 embodiment.

FIG. 5 illustrates use of pushed dynamically loadable data to facilitate a pay-per-use environment.

FIG. 6 illustrates a suitable computing environment in which certain aspects of the invention may be implemented.

Detailed Description

Modern programming languages provide for dynamically loadable, or modular, program code For example, an object-oriented programming (OOP) environment is one in which a program designer defines not only the data type of a data structure, but also the types of operations (e.g., functions/procedures) that can be applied to the data structure. The data structure for a class becomes an object including both data and

functions/procedures. A "class" is a category of such objects, and in modern OOP environments, to avoid unnecessary resource consumption, a class can be dynamically loaded and unloaded to conserve resources.

As will be discussed below, dynamic loading can be effected over a push-type of networking environment. For example, identification data of a class can be announced according to the Service Advertising Protocol (SAP) / Session Description Protocol (SDP) protocol, and the class definition broadcast accordingly. Alternatively, a manifest can be defined that contains identifying data for the definition of a class, and a schedule indicating when a second push will be made that contains the actual program code for the identified class. A programming environment and/or application program can then use a custom class loader to integrate, when needed, pushed classes.

In this description and claims that follow, the term manifest is intended to include both transmission of a manifest, or a SAP/SDP type of announcement of its contents.

FIG. 2 illustrates a variation on the FIG. 1 push environment, in which the network **102** includes wireless networks. Wireless networks include short-range wireless networks including wireless LAN bridges/routers, Bluetooth (a standard promoted by 3Com, Ericsson, Intel, IBM, Lucent, Microsoft, Motorola, Nokia, and Toshiba), or the like, and long-range wireless networks such as microwave systems, satellite systems, cellular communication systems, television transmission towers, etc.

In one embodiment, wireless transmitters comprise a complete transport data pathway between a transmission tower **204** and a receiver, such as a set top box **208**, personal digital assistant, portable computer, handheld computer, wireless appliance, or

other receiving device. In another embodiment, the wireless signal is partially carried on a physical medium, such as a wire, fiber optic, or other medium, before being received by the set top box **208**. Network **206** may or may not be entirely wireless.

Assuming the wireless transmission is an audio and/or visual signal, such as a television signal, a transmitting device **200**, which may be a computer or other device providing television signals, transmits multiple television data streams to a multiplexer **202**. For example, in digital television, Moving Picture Experts Group (MPEG)-2 data transport streams would be emitted from the multiplexer. A transport stream comprises multiple elementary streams of audio, video, and/or other "data" that is multiplexed and marked with a packet identifier (PID). Under the United States Advanced Television Systems Committee (ATSC) digital television standard, a single "channel" is allotted 19.2 Mbps bandwidth. A single television program typically comprises elementary data streams for video and each supported audio language. With data compression, the program may not consume all available bandwidth for the channel, so the program may become one of several "virtual channels" embedded within the physical spectrum allotted to a broadcaster.

The multiplexer combines these multiple data streams into a single transmission that is transmitted by a tower **204**. This transmission is sent over a network **206**, and received by a set top box **208**, such as a cable television decoding-box, computer device with a television decoder, or other television-aware device.

FIG. 3 is a flowchart, according to one embodiment of the invention, for pushing programming code, such as dynamically loadable program code, over a unidirectional communication link such as illustrated in FIG. 1 or FIG. 2.

The unidirectional communication link may be physically unidirectional, as in a transmission from a broadcast tower, or logically unidirectional, as in a unidirectional communication protocol. It is assumed herein that programming code for Java classes is pushed to a receiver. However, the disclosed principles and techniques are equally applicable to Objective-C, C++, SmallTalk, Modula-3, Component Object Model (COM), and other object oriented programming languages and environments, and the invention is not limited in this respect. It will be appreciated by one skilled in the art that other data structures, besides classes, may be pushed as well. For example, a Dynamic Link Library (DLL), or equivalent, having known entry points and structure may be used.

In an object oriented programming environment such as Java, a class encapsulates data, methods and procedures that operate on data input to an instance of the class, and produce appropriate output. Classes are usually collected into libraries for solving particular problems. Java libraries are called "Java archives," or JAR files. To minimize memory requirements and facilitate a more dynamic programming model, Java execution environments defer loading a class into system memory until the class is utilized by an executing application program.

When a class is referenced, if it is not already loaded, it is dynamically loaded and made available to an executing program. Loading requires that a standard location (or locations) be searched for the references class, e.g., to locate a JAR file or other storage of class definition. In Java, an environment variable (or equivalent) called

"CLASSPATH" is expected to exist and indicate a search path for locating class definitions. For example, CLASSPATH may point to directories/folders containing class definitions and/or JAR files, or directly reference a data file storing archives therein. If a class cannot be found and loaded after searching the CLASSPATH environment, loading fails and the corresponding call within the application program fails.

Thus, in one embodiment, to load dynamically loadable program code over a push-type networking environment, a manifest is first prepared **300** corresponding to the dynamically loadable code. The manifest comprises an identifier **302** that identifies the class definition so that the class can be properly loaded during execution of an application program. In Java, the class definition identifier comprises a package name followed by a relative class name. For example, the "String" class is part of the "java.lang" package, and is therefore properly identified as "java.lang.String". Other programming environments may utilize other identifying data, such as the name of the class, and/or a globally unique identifier (GUID) for the class, and/or a class context, and/or class dependencies.

In one embodiment, the manifest further comprises a push schedule **304**, or availability schedule, indicating when class definitions referenced within the manifest will be pushed onto a unidirectional communication link. In this embodiment, the manifest may further comprise a retrieval source **306** if class definitions may be received on one of several unidirectional communication pathways. Other related data **308** may also be stored in the manifest to facilitate routing, verification, billing, or manifest related transactions.

In another embodiment, rather than utilizing a manifest, instead its contents may be directly transmitted over the unidirectional communication link. In this embodiment, an announcement may be broadcast using the SAP/SDP, or equivalent, which indicates an identifier and description of a session, e.g., data to identify dynamically loadable programming code to be pushed, a location from which the code can be retrieved, e.g., a multicast addresses / port for the session, and a schedule of when the code will be pushed.

After creating the manifest, it is pushed **310** onto a unidirectional communication link. Then, in accordance with the push schedule, the dynamically loadable code (e.g., class) referenced by the manifest is pushed **312** onto the unidirectional communication link.

FIG. 4 illustrates a receiver receiving pushed data in accordance with the FIG. 3 embodiment. As illustrated, the receiver listens **400** for, and receives **402**, a pushed manifest. The manifest is parsed **404** to determine **406** identifiers for dynamically loadable code referenced within the manifest. These identifiers are stored **408** within a local memory, such as volatile or non-volatile storage. As discussed above, the manifest contents may be directly transmitted with SAP/SDP type announcements, in which case the receiver listens for announcements to determine identifiers for dynamically loadable code.

A Java application program may be executed **410**. When it references **412** a class for the first time, a search **414** may be performed to locate the class, e.g., the CLASSPATH environment is searched. Assuming searching the CLASSPATH fails, a

search **416** may be performed to locate the class among pushed class identifiers stored **408** within the local memory. If an appropriate class identifier(s) is located, the schedule corresponding to the identifier may be inspected **418**. Dynamically loadable programming code, e.g., a Java class file, may then be retrieved **420** from a unidirectional communication link according to the schedule. If SAP/SDP type broadcasting is in use, the dynamically loadable programming code may be retrieved from the multicast address / port indicated in the session's announcement.

In one embodiment, the retrieved dynamically loadable code may be added to the existing CLASSPATH environment so as to avoid future latency in utilizing the retrieved code. In one embodiment, the CLASSPATH comprises a combination of volatile and nonvolatile storage. In this embodiment, standard classes may be stored in nonvolatile storage, such as (erasable) programmable read-only memory (PROM), non-volatile random access memory (NVRAM), complementary metal oxide semiconductor (CMOS), hard-drives, memory sticks, etc. Retrieved code, however, may be stored in volatile storage, e.g., random access memory (RAM).

Thus, for example, portable electronics, set-top boxes, or other electronics not configured with mass storage may be shipped with standard class definitions, and temporarily acquire auxiliary classes as needed during execution of an application program.

After retrieval **420**, the dynamically loadable code is loaded **422** in a manner customary to the program language environment in use for execution **424** by the application program. Note that loading **422** may be performed asynchronously to execution of the application program causing the retrieval.

In addition, retrieval and loading may be performed transparently by an operating system, programming environment, or the application program depending on receiver environment configuration. For example, a Java runtime environment may determine a class needs to be retrieved **420** from a unidirectional communication link, and then load the class **422** without an application program becoming aware of the retrieval process.

FIG. 5 illustrates use of pushed dynamically loadable data to facilitate a pay-per-use environment. Typical pay-per-use environments include television, telephone, or computer network broadcast systems, where one or more "premium" data channels are encrypted and transmitted to a receiver, and the receiver can not use the encrypted data without obtaining a decrypting ability.

In this illustrated embodiment, pushed manifests discussed above include identifiers for decryption classes that allow a receiving device to permanently or temporarily obtain encryption ability. The period of availability for decryption capability may be specified in the manifest, or regulated by local and/or remote policies. In one embodiment, temporary encryption is achieved by storing decryption classes in volatile memory that is cleared when a receiving device is reset, power cycled, etc. Clearance may also be according to a clearance schedule, such as hourly, daily, weekly, etc., or according to a schedule indicated within a manifest, e.g., automatic clearance after a certain number of uses, or after an elapsed time after first use, etc.

Thus, a computing device, such as a set top box or portable electronic receiver, e.g., portable television, personal digital assistant, cellular telephone, portable computer, wireless appliance, or the like, executes **500** a control program, such as an

operating system or application program. The control program receives **502** a manifest that includes an identifier and push schedule for a decryption class, as well as purchasing data indicating a cost associated with obtaining the premium data by way of the decryption class. A user of the device elects **504**, e.g., indicates by way of a graphical user interface, selection of a button on the computing device, etc., a desire to purchase the premium data in accordance with the purchasing data. In response, an appropriate purchase transaction occurs **506** according to the purchasing data in the manifest.

Note that purchase transactions are intended to include purchasing protocols making use of third-party processing and/or billing arrangements. Since the communication link is unidirectional, it is assumed an alternative communication arrangement exists to effect payment. For example, the computing device may have an internal modem in communication with a telephone service, a network interface in communication with a network, or other communication pathway. Purchases may be made immediately, or performed on a delayed basis, e.g., the computing device receives election **504** of a purchase, and decryption is performed on assumption that payment will eventually be secured. Delayed payment facilitates using portable receivers intermittently connected to an alternative communication arrangement.

After the purchase transaction **506**, the computing device retrieves **508** an appropriate decryption class from the unidirectional communication link in accordance with the push schedule. The decryption class may be installed **510** as indicated by the manifest, e.g., it may be installed either permanently or temporarily. In a Java-type execution environment, the decryption class is installed within the CLASSPATH

environment. The decryption class is then executed **512** by the control program to decrypt desired data.

In one embodiment, to prevent theft of the decryption class, the decryption class is itself encoded within a private key of a public/private key pairing in a public key cryptosystem. The communication device may be configured to contain the public key of the pairing, and the communication device decrypts the decryption class with the public key only after purchase transaction **506**. In such fashion, decryption classes can be blindly pushed onto an unsecured unidirectional communication link without regard to illicit theft of services.

FIG. 6 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which certain aspects of the illustrated invention may be implemented.

An exemplary system for implementing the invention includes a computing device **600** having system bus **602** for coupling various computing device components. Typically, attached to the bus are non-programmable and programmable processors **604**, a memory **606** (e.g., RAM, ROM), storage devices **608**, a video interface **610**, and input/output interface ports **612**. Storage devices include hard-drives, floppy-disks, optical storage, magnetic cassettes, tapes, flash memory cards, memory sticks, digital video disks, and the like.

The invention may be described by reference to different high-level program modules and/or low-level hardware contexts. Those skilled in the art will realize that program modules can be interchanged with low-level hardware instructions. Program

modules include procedures, functions, programs, components, data structures, and the like, for performing particular tasks or implementing particular abstract data types. Modules may be incorporated into single and multi-processor computing devices, Personal Digital Assistants (PDAs), cellular telephones, portable computers, handheld computers, wireless appliances, and the like. Thus, the storage systems and associated media can store data and executable instructions for the computing device.

The computing device is expected to operate in a networked environment using logical connections to one or more remote computing devices **614**, **616** through a network interface **618**, modem **620**, or other communication pathway. Computing devices may be interconnected by way of a network **622** such as an intranet, the Internet, or other network. Modules may be implemented within a single computing device, or processed in a distributed network environment, and stored in both local and remote memory. Thus, for example, with respect to the illustrated embodiments, assuming computing device **600** is a transmitter of pushed dynamically loadable programming code, then remote devices **614**, **616** may respectively be a set top box and a portable wireless receiver of the pushed code.

It will be appreciated that remote computing devices **614**, **616** may be configured like computing device **600**, and therefore include many or all of the elements discussed for computing device. It should also be appreciated that computing devices **600**, **614**, **616** may be embodied within a single device, or separate communicatively-coupled components, and may include or be embodied within routers, bridges, peer devices, web servers, and application programs utilizing network application protocols such as the HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), and the like.

Having described and illustrated the principles of the invention with reference to illustrated embodiments, it will be recognized that the illustrated embodiments can be modified in arrangement and detail without departing from such principles.

And, even though the foregoing discussion has focused on particular embodiments, it is understood that other configurations are contemplated. In particular, even though expressions such as “in one embodiment,” “in another embodiment,” or the like are used herein, these phrases are meant to generally reference embodiment possibilities, and are not intended to limit the invention to particular embodiment configurations. As used herein, these terms may reference the same or different embodiments, and unless implicitly or expressly indicated otherwise, embodiments are combinable into other embodiments. Consequently, in view of the wide variety of permutations to the above-described embodiments, the detailed description is intended to be illustrative only, and should not be taken as limiting the scope of the invention.

What is claimed as the invention, therefore, is all such modifications as may come within the scope and spirit of the following claims and equivalents thereto.